Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Для организации иерархического построения объектов необходимо разработать базовый класс, который содержит функционал и свойства для построения иерархии объектов. В последующем, в приложениях использовать этот класс как базовый для всех создаваемых классов. Это позволит включать любой объект в состав дерева иерархии объектов.

Каждый объект на дереве иерархии имеет свое место и наименование. Не допускается для одного головного объекта одинаковые наименования в составе подчиненных объектов.

Создать базовый класс со следующими элементами:

- Свойства:
  - Наименование объекта (строкового типа);
  - Указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно nullptr);
  - Динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.
- Функционал:
  - Параметризированный конструктор с параметрами: указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии; строкового типа, содержащий наименование создаваемого объекта (имеет значение по умолчанию);
  - Метод редактирования имени объекта. Один параметр строкового типа, содержит новое наименование объекта. Если нет дубляжа имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь»;
  - Метод получения имени объекта;

- Метод получения указателя на головной объект текущего объекта;
- Метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- Метод получения указателя на непосредственно подчиненный объект по его имени. Если объект не найден, то возвращает nullptr. Один параметр строкового типа, содержит наименование искомого подчиненного объекта.

Для построения дерева иерархии объектов в качестве корневого объекта используется объект приложение. Класс объекта приложения наследуется от базового класса. Объект приложение реализует следующий функционал:

- Метод построения исходного дерева иерархии объектов (конструирования моделируемой системы);
- Метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Написать программу, которая последовательно строит дерево иерархии объектов, слева направо и сверху вниз. Переход на новый уровень происходит только от правого (последнего) объекта предыдущего уровня. Для построения дерева использовать объекты двух производных классов, наследуемых от базового. Исключить создание объекта если его наименование совпадает с именем уже имеющегося подчиненного объекта у предполагаемого головного. Исключить добавление нового объекта, не последнему подчиненному предыдущего уровня.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Основная функция должна иметь следующий вид:

```
int main ( )
{
    cl_application  ob_cl_application ( nullptr ); // создание корневого объекта
     ob_cl_application.build_tree_objects ( );         // конструирование системы, построение дерева объектов
    return ob_cl_application.exec_app ( );         // запуск системы
```

```
}
```

Наименование класса cl_application и идентификатора корневого объекта ob_cl_application могут быть изменены разработчиком.

Все версии курсовой работы имеют такую основную функцию.

1.1 Описание входных данных

Первая строка:

```
«имя корневого объекта»
```

Вторая строка и последующие строки:

```
«имя головного объекта» «имя подчиненного объекта»
```

Создается подчиненный объект и добавляется в иерархическое дерево. Если «имя головного объекта» равняется «имени подчиненного объекта», то новый объект не создается и построение дерева объектов завершается.

Пример ввода

```
Object_root
Object_root Object_1
Object_root Object_2
Object_root Object_3
Object_3 Object_4
Object_3 Object_5
Object_6 Object_6
```

Дерево объектов, которое будет построено по данному примеру:

```
Object_root
    Object_1
    Object_2
    Object_3
        Object_4
        Object_5
```

1.2 Описание выходных данных

Первая строка:

«имя корневого объекта»

Вторая строка и последующие строки имена головного и подчиненных объектов очередного уровня разделенных двумя пробелами.

«имя головного объекта» «имя подчиненного объекта»[[«имя подчиненного объекта»] …….]

Пример вывода:

```
Object_root
Object_root  Object_1  Object_2  Object_3
Object_3  Object_4  Object_5
```

2 МЕТОД РЕШЕНИЯ

Объект стандартного потока ввода и вывода (cin, cout)

Условный оператор if

Оператор со счетчиком for

Оператор цикла с предусловием while

Класс cl_base

- Поля
  - s_name - тип string, наименование объекта, модификатор доступа private
  - p_head_object- тип cl_base, указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно nullptr), модификатор доступа private
  - p_sub_object- тип cl_base, динамический массив (вектор) указателей на объекты, подчиненные к текущему объекту в дереве иерархии, модификатор доступа private
- методы
  - Параметризированный конструктор cl_base (cl_base* p_head_object, string s_name) - присваивает полю p_head_object значение аргумента p_head_object, а полю s_name значение аргумента s_name, модификатор доступа public
  - bool edit_name(string name) - содержит новое наименование объекта. Если нет дубляжа имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь», модификатор доступа public
  - bool set_name() - редактирования имени объекта, модификатор доступа public

- string get_name() - возвращает имени объекта, модификатор доступа public
- cl_base* get_head_object() - возвращает указатель на головной объект текущего объекта, модификатор доступа public
- void print_tree() - метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз, модификатор доступа public

Класс cl_application:

- Методы:
  - Метод build_tree_objects () - построения исходного дерева иерархии объектов, модификатор доступа public
  - Метод exec_app() - запуска приложения модификатор доступа public
  - Параметризированный конструктор с параметрами cl_application(cl_base* p_head_object) : указатель на объект базового класса, содержащий адрес головного объекта вдереве иерархии; строкового типа, содержащий наименование создаваемого объекта, который вызывает конструктор cl_base

Класс cl_1:

- Параметризированный конструктор с параметрами cl_1(cl_base* p_head, string s_name): указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии; строкового типа, содержащий наименование создаваемого объекта, который вызывает конструктор cl_base

Таблица 1 – Иерархия наследования классов

№	Наименование	Классы наследник	Модификатор	Описание	Номер классов	Комментарий

		и	доступа		наследников	
1	cl_base			базовый класс		
		cl_application	public		2	
		cl_1	public		3	
2	cl_application			производный класс от cl_base		
3	cl_1			производный класс от cl_base		

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм конструктора класса cl_base

Функционал: добавление в состав подчиненных головного объекта.

Параметры: p_head_object - указатель на объект базового класса; s_name - строкового типа.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		присвоение указателю на головной объект значение аргумента p_head_object	2
2		Полю s_name текущего объекта присваиваем значение s_name	3
3	Значение параметра p_head_object != nullptr	добавление в конец контейнера типа vector p_sub_objects(поле) головного объекта для текущего объекта указателя на текущий объект те	∅
			∅

3.2 Алгоритм метода edit_name класса cl_base

Функционал: содержит новое наименование объекта. Если нет дубляжа

имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь».

Параметры: string new_name.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода edit_name класса cl_base

№	Предикат	Действия	№ перехода
1	cl_base* child: p_head_object-> children		2
			3
2	child != this && child->name == new_name	возврат false	∅
			1
3		присваение полю s_name значение аргумента s_new_name	4
4		возврат true	∅

3.3 Алгоритм метода get_sub_object класса cl_base

Функционал: получения указателя на непосредственно подчиненный объект по его имени.

Параметры: нет .

Возвращаемое значение: cl_base* p_sub_object.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_sub_object класса cl_base

№	Предикат	Действия	№ перехода
1		иницилизация переменной i = 0 типа int	2
2	i<p_sub_objects.size	i++	3

№	Предикат	Действия	№ перехода
			∅
3	this->p_sub_objects[i]->s_name==s_name	возвращает указатель на p_sub_objects[i]	∅
		возвращает nullptr	∅

3.4 Алгоритм метода get_head_object класса cl_base

Функционал: возвращает указатель на головной объект текущего объекта.

Параметры: нет.

Возвращаемое значение: cl_base* p_head_object.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_head_object класса cl_base

№	Предикат	Действия	№ перехода
1		возвращает значение поля p_head_object	∅

3.5 Алгоритм метода print_tree класса cl_base

Функционал: вывод иерархии.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1	p_sub_objects.size() == 0		∅

№	Предикат	Действия	№ перехода
		вывод значение s_name	2
2		инициализация переменной i = 0 типа int	3
3	i<p_sub_objects.size	вывод значение p_sub_objects	4
			5
4		i++	3
5	i<p_sub_objects.size	p_sub_objects вызывает метод print_tree	6
			∅
6		i++	5

3.6 Алгоритм метода set_name класса cl_base

Функционал: редактирования имени объекта.

Параметры: string s_new_name.

Возвращаемое значение: bool True||False.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода set_name класса cl_base

№	Предикат	Действия	№ перехода
1	Указатель на головной объект не является пустым		2
			6
2		инициализация переменной i = 0 типа int	3
3	i<колличеству подчиненных объектов		4
			7

№	Предикат	Действия	№ перехода
4		i++	5
5	имя подчиненного объекта равен новому значению	возврат False	∅
			3
6		имени подчиненного объекта присваивается новое значение	7
7		возврат True	∅

3.7 Алгоритм деструктора класса cl_base

Функционал: очистка памяти.

Параметры: нет .

Алгоритм деструктора представлен в таблице 8.

Таблица 8 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		иницилизация переменной i = 0 типа int	2
2	i<p_sub_objects.size	удаление элементов из p_sub_objects	3
			∅
3		i++	∅

3.8 Алгоритм метода exec_app класса cl_application

Функционал: запуска приложения.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		вывод s_name этого объекта	2
2		вызов метода print_tree этого объекта	∅

3.9 Алгоритм метода build_tree_objects класса cl_application

Функционал: построения исходного дерева иерархии объектов.

Параметры: нет .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		объявление переменных s_head_name, s_sub_name строкового типа	2
2		cl_base* p_head= this; cl_base* p_sub= nullptr;	3
3	true	ввод s_head_name, s_sub_name	4
			∅
4	имя головного объекта равен s_sub_name	выход	∅
			5
5	указатель на p_sub не равен пустому множеству и имя головного объекта равен	p_head=p_sub	6

№	Предикат	Действия	№ перехода
	имени p_sub		
			6
6	подчиненный объект не равен пустому множеству	p_sub = new cl_1(p_head, s_sub_name)	3
			∅

3.10 Алгоритм конструктора класса cl_application

Функционал: вызов конструктор класса cl_base.

Параметры: cl_base* p_head_object, string s_name.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода
1		вызов конструктор класса cl_base	∅

3.11 Алгоритм конструктора класса cl_1

Функционал: вызов конструктор класса cl_base.

Параметры: cl_base* p_head_object, string s_name.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl_1

№	Предикат	Действия	№ перехода
1		вызов конструктор класса cl_base	∅

3.12 Алгоритм функции main

Функционал: основная программа.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 13.

Таблица 13 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание корневого объекта	2
2		Конструирование системы, построение дерева	3
3		запуск системы	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-8.

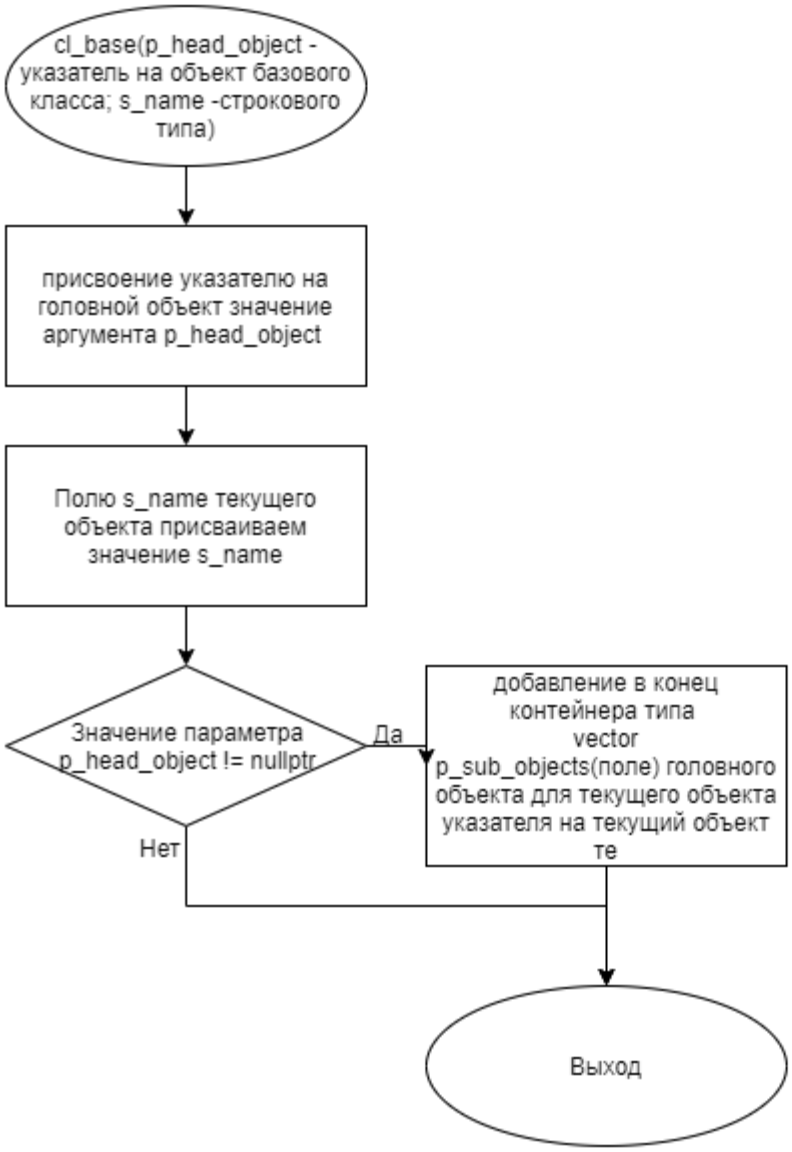

Рисунок 1 – Блок-схема алгоритма

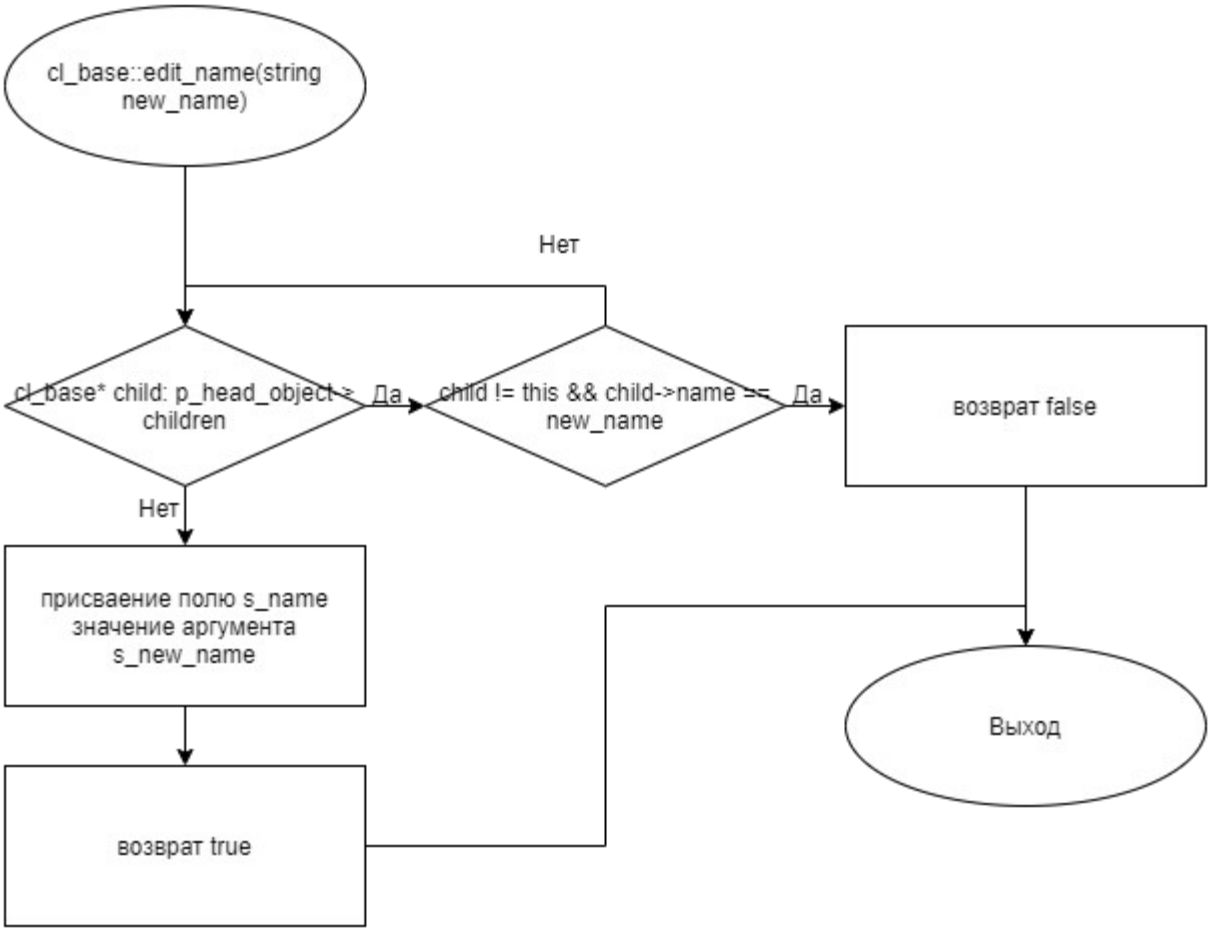

Рисунок 2 – Блок-схема алгоритма

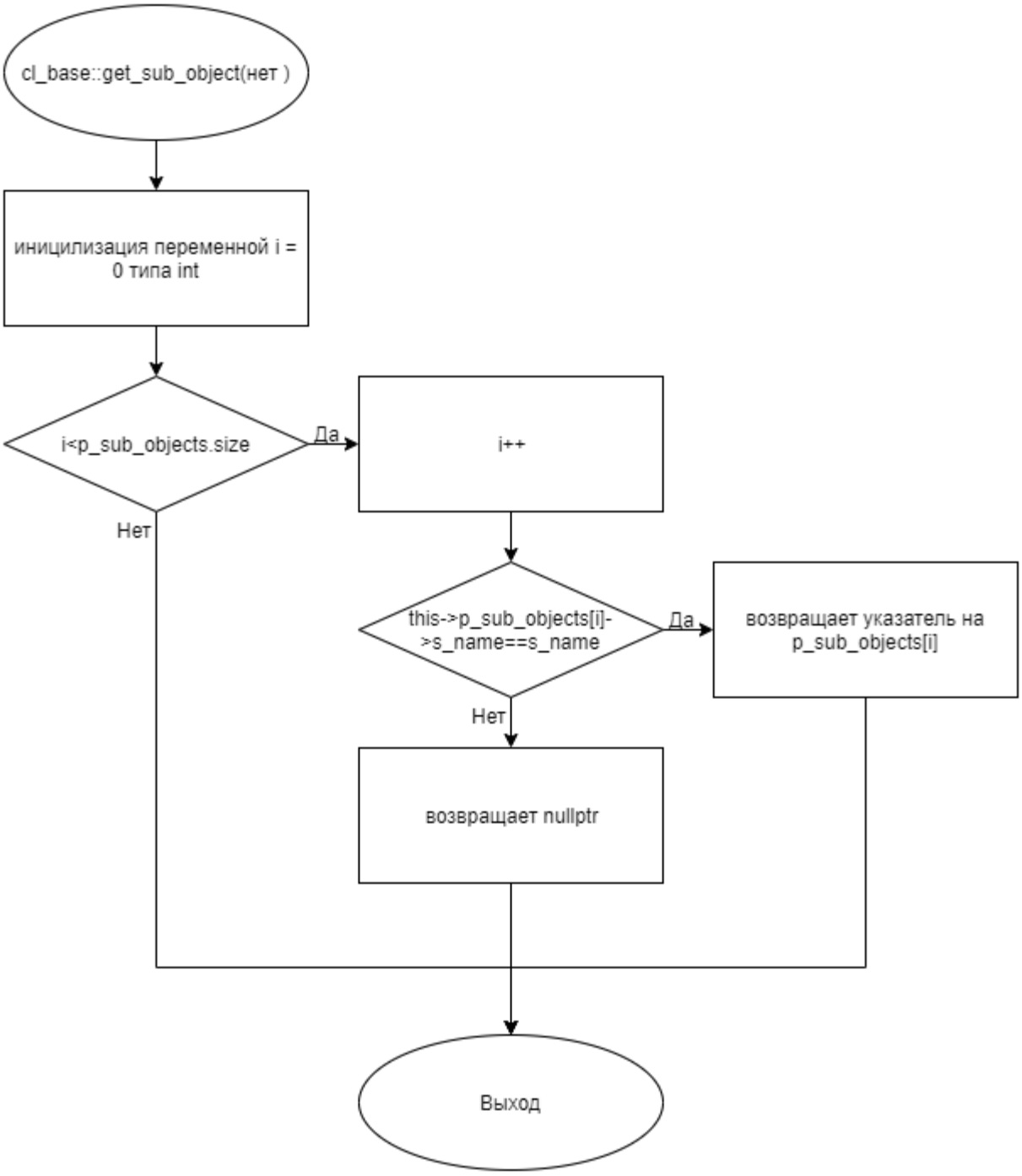

Рисунок 3 – Блок-схема алгоритма

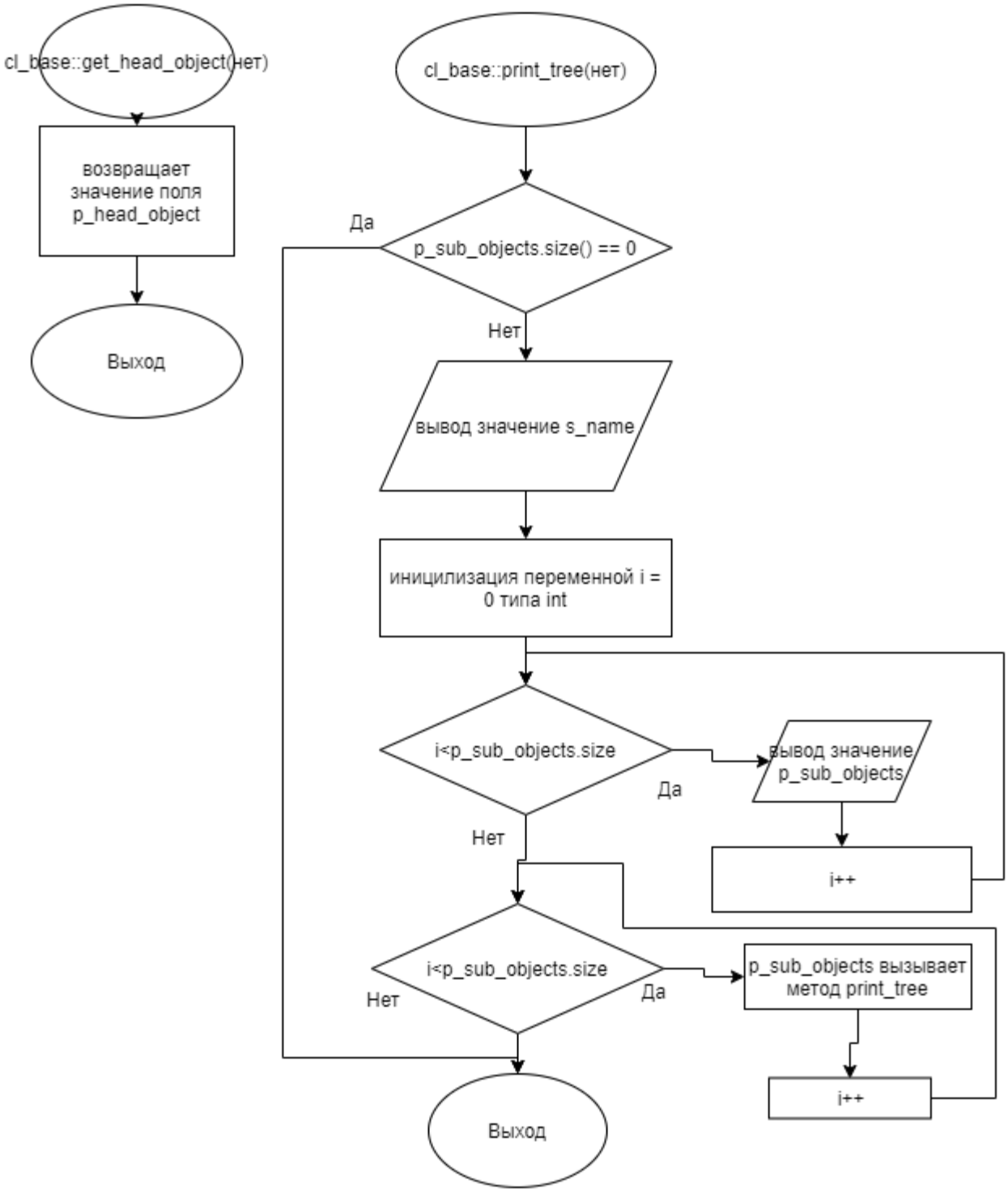

Рисунок 4 – Блок-схема алгоритма

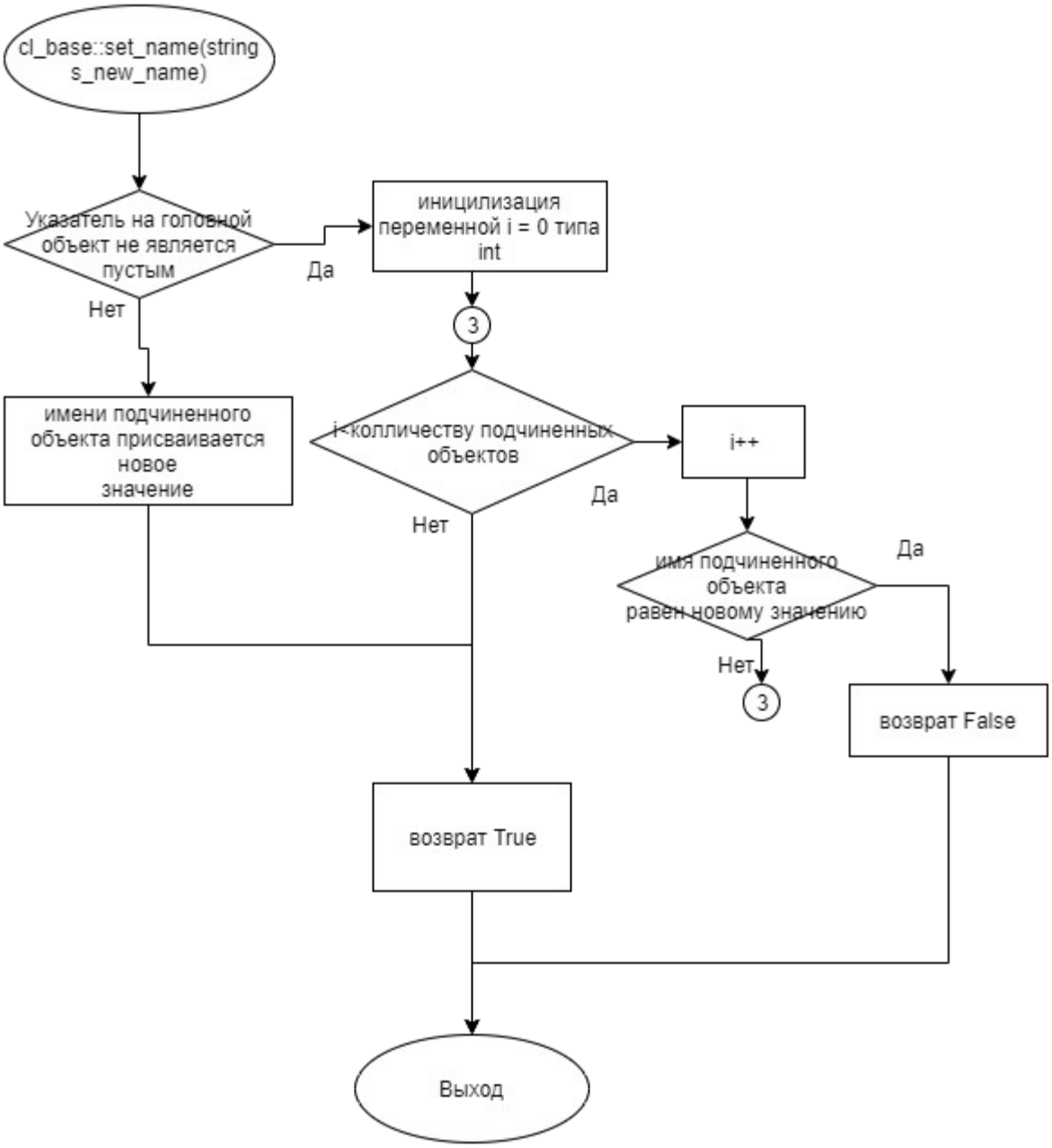

Рисунок 5 – Блок-схема алгоритма

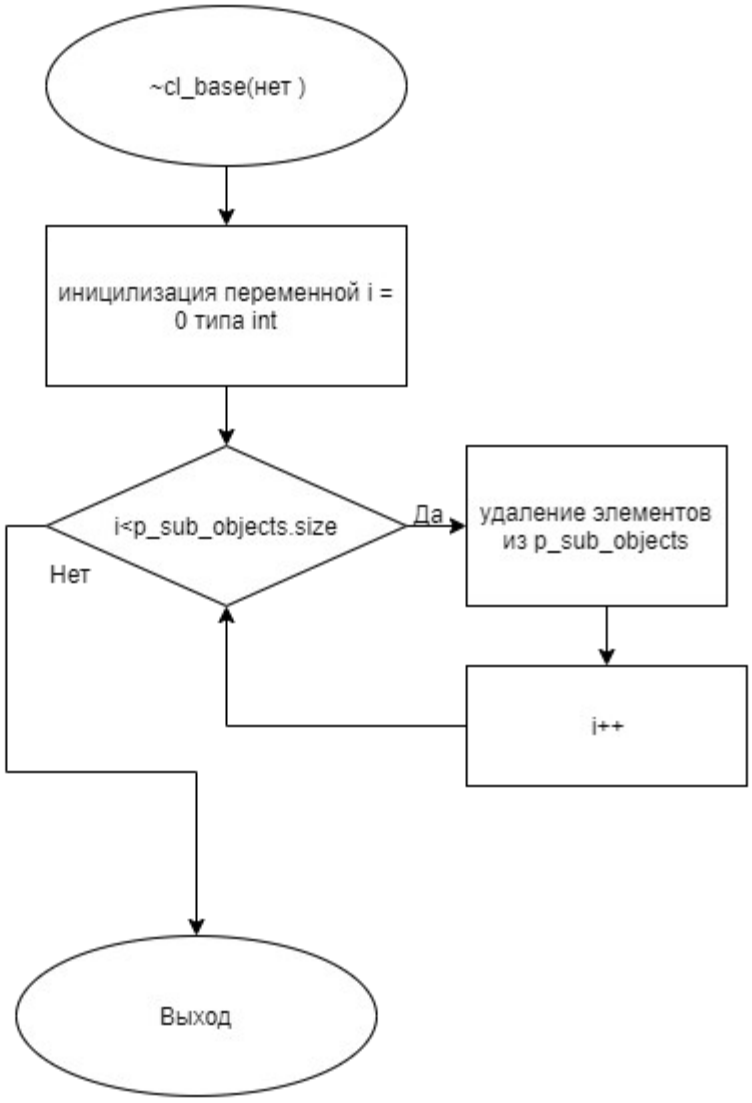

Рисунок 6 – Блок-схема алгоритма

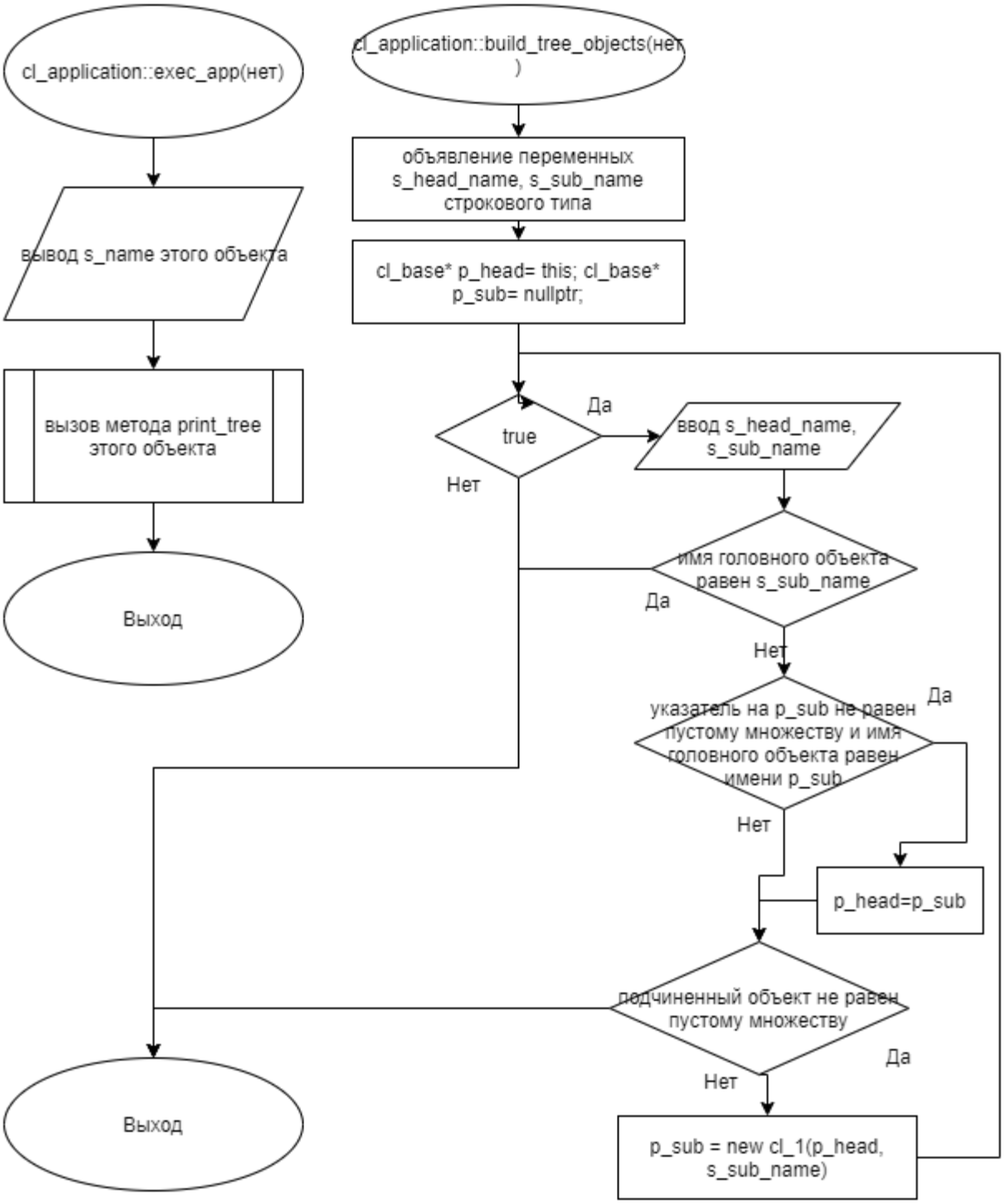

Рисунок 7 – Блок-схема алгоритма

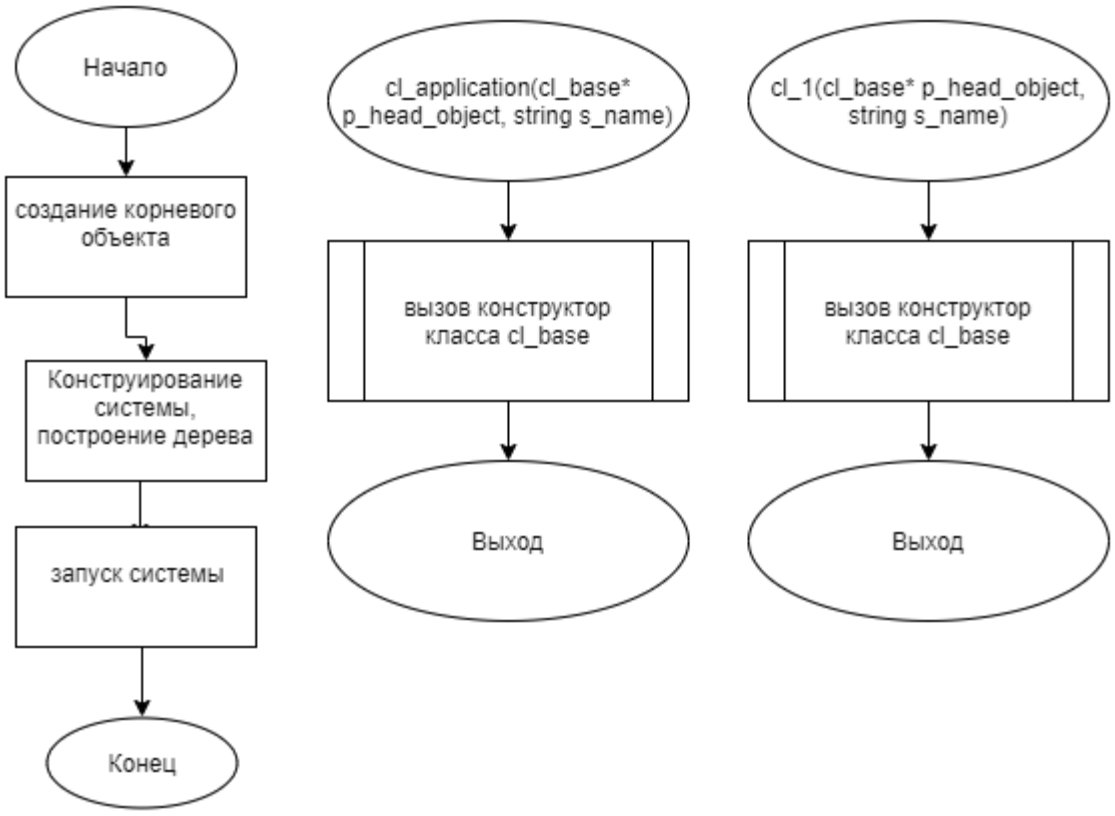

Рисунок 8 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"
#include <iostream>
using namespace std;

cl_1::cl_1(cl_base* p_head, string s_name):cl_base(p_head, s_name){};
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef CL_1__H
#define CL_1__H
#include "cl_base.h"
#include <iostream>
using namespace std;

class cl_1: public cl_base{
      public:
            cl_1(cl_base* p_head, string s_name);
};
#endif
```

5.3 Файл cl_application.cpp

Листинг 3 – cl_application.cpp

```
#include "cl_application.h"
#include <iostream>
using namespace std;

cl_application::cl_application(cl_base* p_head_object): cl_base(p_head_object){};

void cl_application:: build_tree_objects(){
```

```
	string s_head_name, s_sub_name;
	cl_base* p_head = this;
	cl_base* p_sub = nullptr;
	cl_base* last_sub = nullptr;
	cin>> s_head_name;
	this->set_name(s_head_name);
	while (true){
		cin>> s_head_name>> s_sub_name;
		if (s_head_name==s_sub_name){
			break;
		}
		if (p_sub!=nullptr && s_head_name== p_sub->get_name()){
			p_head = p_sub;
		}
		if  (p_head->get_sub_object(s_sub_name)==nullptr  &&  s_head_name  ==
p_head-> get_name()){
			p_sub = new cl_1(p_head, s_sub_name);

		}




	}
}
int cl_application::exec_app() {
	cout << this->get_name();
	this->print_tree();
	return 0 ;
}
```

5.4 Файл cl_application.h

Листинг 4 – cl_application.h

```
#ifndef CL_APPLICATION__H
#define CL_APPLICATOIN__H
#include <iostream>
#include "cl_base.h"
#include "cl_1.h"
using namespace std;

class cl_application: public cl_base{
	public:
		cl_application(cl_base* p_head_object);
		void build_tree_objects();
		int exec_app();
};

#endif
```

5.5 Файл cl_base.cpp

Листинг 5 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base* p_head_object, string s_name){
this->p_head_object = p_head_object;
this->s_name = s_name;
if(this->p_head_object != nullptr) {
      p_head_object  ->  p_sub_objects.push_back(this);  //  добавление  в  состав
подчиненных головного объекта
      }
}

bool cl_base::set_name(string s_new_name){
      if(p_head_object != nullptr) {
            for (int i = 0; i < p_head_object->p_sub_objects.size(); i++){
                  if (p_head_object->p_sub_objects[i]->s_name== s_new_name) {
                        return false;
                  }
            }
      }
      s_name = s_new_name;
      return true;
}

string cl_base::get_name(){
      return this->s_name;
}

cl_base* cl_base::get_head_object(){
      return this->p_head_object;
}

void cl_base:: print_tree(){
      if (this->p_sub_objects.size()==0) {
            return;
      }
      else {
            cout << endl<< s_name;
            for (int i = 0; i < p_sub_objects.size(); i++){//
                  cout << "  "<< this->p_sub_objects[i]->s_name;
            }
            for (int i=0 ; i < this->p_sub_objects.size(); i++) {
                  this -> p_sub_objects[i]->print_tree();
            }
      }
}

cl_base::~cl_base(){
      for (int i=0 ; i < this->p_sub_objects.size(); i++) {
                  delete p_sub_objects[i];
            }
}
```

```
cl_base* cl_base::get_sub_object(string S_name){
	for (int i=0 ; i < this->p_sub_objects.size(); i++) {
			if (this->p_sub_objects[i]-> s_name == S_name) {
				return p_sub_objects[i];
			}
		}
	return nullptr;
}
```

5.6 Файл cl_base.h

Листинг 6 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <vector>
#include <string>
#include <iostream>
using namespace std;

class cl_base{
	string s_name;
	cl_base* p_head_object;
	vector <cl_base*> p_sub_objects;
	public:
	cl_base(cl_base* p_head_object, string s_name= "Base object");
	bool set_name(string s_new_name);
	string get_name();
	cl_base* get_head_object();
	void print_tree();
	~cl_base();
	cl_base* get_sub_object(string S_name);
};
#endif
```

5.7 Файл main.cpp

Листинг 7 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "cl_application.h"
#include <iostream>
using namespace std;

int main()
{
	cl_application ob_cl_application(nullptr);
```

```
        ob_cl_application.build_tree_objects();
        return ob_cl_application.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 14.

Таблица 14 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
object_root object_root object_1 object_root object_2 object_root object_3 object_3 object_4 object_3 object_5 object_6 object_6	object_root object_root object_1 object_2 object_3 object_3 object_4 object_5	object_root object_root object_1 object_2 object_3 object_3 object_4 object_5

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).